三权分立的CKM矩阵

三权分立的定量研究

May 2023

About this presentation

This presentation is supposed to briefly showcase what you can do with this template.

For a full documentation, read the online book.

A title

Let's explore what we have here.

On the top of this slide, you can see the slide title.

We used the `title` argument of the `#slide` function for that:

```
#slide(title: "First slide")[
    ...
]
```

Titles are not mandatory, this slide doesn't have one.

But did you notice that the current section name is displayed above that top line?

We defined it using `#new-section("Introduction")`.

This helps our audience with not getting lost after a microsleep.

The bottom of the slide

Now, look down!

There we have some general info for the audience about what talk they are actually attending right now.

You can also see the slide number there.

A dynamic slide with pauses

Sometimes we don't want to display everything at once.

A dynamic slide with pauses

Sometimes we don't want to display everything at once.

That's what the `pause` function is there for! Use it as

```
#show: pause(n)
```

A dynamic slide with pauses

Sometimes we don't want to display everything at once.

That's what the `pause` function is there for! Use it as

```
#show: pause(n)
```

It makes everything after it appear at the `n`–th subslide.

(Also note that the slide number does not change while we are here.)

Fine-grained control

When `#pause` does not suffice, you can use more advanced commands to show or hide content.

These are your options:

- `#uncover`
- `#only`
- `#alternatives`
- `#one-by-one`
- `#line-by-line`

Let's explore them in more detail!

#uncover: Reserving space

With `#uncover`, content still occupies space, even when it is not displayed.

For example, are only visible on the second "subslide".

In `()` behind `#uncover`, you specify when to show the content, and in `[]` you then say what to show:

```
#uncover(3)[Only visible on the third "subslide"]
```

#uncover: Reserving space

With `#uncover`, content still occupies space, even when it is not displayed.

For example, these words are only visible on the second “subslide”.

In `()` behind `#uncover`, you specify when to show the content, and in `[]` you then say what to show:

```
#uncover(3)[Only visible on the third "subslide"]
```

#uncover: Reserving space

With `#uncover`, content still occupies space, even when it is not displayed.

For example, are only visible on the second “subslide”.

In `()` behind `#uncover`, you specify when to show the content, and in `[]` you then say what to show:

```
#uncover(3)[Only visible on the third "subslide"]
```

Only visible on the third “subslide"

Complex display rules

So far, we only used single subslide indices to define when to show something.

We can also use arrays of numbers...

```
#uncover((1, 3, 4))[Visible on subslides 1, 3, and 4]
```

Visible on subslides 1, 3, and 4

...or a dictionary with `beginning` and/or `until` keys:

```
#uncover((beginning: 2, until: 4))[Visible on subslides 2, 3, and 4]
```

Complex display rules

So far, we only used single subslide indices to define when to show something.

We can also use arrays of numbers...

```
#uncover((1, 3, 4))[Visible on subslides 1, 3, and 4]
```

...or a dictionary with `beginning` and/or `until` keys:

```
#uncover((beginning: 2, until: 4))[Visible on subslides 2, 3, and 4]
```

Visible on subslides 2, 3, and 4

Complex display rules

So far, we only used single subslide indices to define when to show something.

We can also use arrays of numbers...

```
#uncover((1, 3, 4))[Visible on subslides 1, 3, and 4]
```

Visible on subslides 1, 3, and 4

...or a dictionary with `beginning` and/or `until` keys:

```
#uncover((beginning: 2, until: 4))[Visible on subslides 2, 3, and 4]
```

Visible on subslides 2, 3, and 4

Complex display rules

So far, we only used single subslide indices to define when to show something.

We can also use arrays of numbers...

```
#uncover((1, 3, 4))[Visible on subslides 1, 3, and 4]
```

Visible on subslides 1, 3, and 4

...or a dictionary with `beginning` and/or `until` keys:

```
#uncover((beginning: 2, until: 4))[Visible on subslides 2, 3, and 4]
```

Visible on subslides 2, 3, and 4

Convenient rules as strings

As as short hand option, you can also specify rules as strings in a special syntax.

Comma separated, you can use rules of the form

`1-3`	from subslide 1 to 3 (inclusive)
`-4`	all the time until subslide 4 (inclusive)
`2-`	from subslide 2 onwards
`3`	only on subslide 3

```
#uncover("-2, 4-6, 8-")[Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards]
```

Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards

Convenient rules as strings

As as short hand option, you can also specify rules as strings in a special syntax.

Comma separated, you can use rules of the form

`1-3`	from subslide 1 to 3 (inclusive)
`-4`	all the time until subslide 4 (inclusive)
`2-`	from subslide 2 onwards
`3`	only on subslide 3

```
#uncover("-2, 4-6, 8-")[Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards]
```

Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards

Convenient rules as strings

As as short hand option, you can also specify rules as strings in a special syntax.

Comma separated, you can use rules of the form

`1-3`	from subslide 1 to 3 (inclusive)
`-4`	all the time until subslide 4 (inclusive)
`2-`	from subslide 2 onwards
`3`	only on subslide 3

```
#uncover("-2, 4-6, 8-")[Visible on subslides 1, 2, 4, 5, 6, and from 8
onwards]
```

Convenient rules as strings

As as short hand option, you can also specify rules as strings in a special syntax.

Comma separated, you can use rules of the form

`1-3`	from subslide 1 to 3 (inclusive)
`-4`	all the time until subslide 4 (inclusive)
`2-`	from subslide 2 onwards
`3`	only on subslide 3

```
#uncover("-2, 4-6, 8-")[Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards]
```

Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards

Convenient rules as strings

As as short hand option, you can also specify rules as strings in a special syntax.

Comma separated, you can use rules of the form

`1-3`	from subslide 1 to 3 (inclusive)
`-4`	all the time until subslide 4 (inclusive)
`2-`	from subslide 2 onwards
`3`	only on subslide 3

```
#uncover("-2, 4-6, 8-")[Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards]
```

Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards

Convenient rules as strings

As as short hand option, you can also specify rules as strings in a special syntax.

Comma separated, you can use rules of the form

`1-3`	from subslide 1 to 3 (inclusive)
`-4`	all the time until subslide 4 (inclusive)
`2-`	from subslide 2 onwards
`3`	only on subslide 3

```
#uncover("-2, 4-6, 8-")[Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards]
```

Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards

Convenient rules as strings

As as short hand option, you can also specify rules as strings in a special syntax.

Comma separated, you can use rules of the form

`1-3`	from subslide 1 to 3 (inclusive)
`-4`	all the time until subslide 4 (inclusive)
`2-`	from subslide 2 onwards
`3`	only on subslide 3

```
#uncover("-2, 4-6, 8-")[Visible on subslides 1, 2, 4, 5, 6, and from 8
onwards]
```

Convenient rules as strings

As as short hand option, you can also specify rules as strings in a special syntax.

Comma separated, you can use rules of the form

`1-3`	from subslide 1 to 3 (inclusive)
`-4`	all the time until subslide 4 (inclusive)
`2-`	from subslide 2 onwards
`3`	only on subslide 3

```
#uncover("-2, 4-6, 8-")[Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards]
```

Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards

#only: Reserving no space

Everything that works with #uncover also works with #only.

However, content is completely gone when it is not displayed.

For example, the rest of this sentence moves.

Again, you can use complex string rules, if you want.

```
#only("2-4, 6")[Visible on subslides 2, 3, 4, and 6]
```

#only: Reserving no space

Everything that works with #uncover also works with #only.

However, content is completely gone when it is not displayed.

For example, see how the rest of this sentence moves.

Again, you can use complex string rules, if you want.

```
#only("2-4, 6")[Visible on subslides 2, 3, 4, and 6]
```

Visible on subslides 2, 3, 4, and 6

#only: Reserving no space

Everything that works with #uncover also works with #only.

However, content is completely gone when it is not displayed.

For example, the rest of this sentence moves.

Again, you can use complex string rules, if you want.

```
#only("2-4, 6")[Visible on subslides 2, 3, 4, and 6]
```

Visible on subslides 2, 3, 4, and 6

#only: Reserving no space

Everything that works with #uncover also works with #only.

However, content is completely gone when it is not displayed.

For example, the rest of this sentence moves.

Again, you can use complex string rules, if you want.

```
#only("2-4, 6")[Visible on subslides 2, 3, 4, and 6]
```

Visible on subslides 2, 3, 4, and 6

#only: Reserving no space

Everything that works with #uncover also works with #only.

However, content is completely gone when it is not displayed.

For example, the rest of this sentence moves.

Again, you can use complex string rules, if you want.

```
#only("2-4, 6")[Visible on subslides 2, 3, 4, and 6]
```

#only: Reserving no space

Everything that works with #uncover also works with #only.

However, content is completely gone when it is not displayed.

For example, the rest of this sentence moves.

Again, you can use complex string rules, if you want.

```
#only("2-4, 6")[Visible on subslides 2, 3, 4, and 6]
```

Visible on subslides 2, 3, 4, and 6

#alternatives: Substituting content

You might be tempted to try

```
#only(1)[Ann] #only(2)[Bob] #only(3)[Christopher] likes #only(1)
[chocolate] #only(2)[strawberry] #only(3)[vanilla] ice cream.
```

Ann likes chocolate ice cream.

But it is hard to see what piece of text actually changes because everything moves around. Better:

```
#alternatives[Ann][Bob][Christopher] likes #alternatives[chocolate]
[strawberry][vanilla] ice cream.
```

Ann likes chocolate ice cream.

#alternatives: Substituting content

You might be tempted to try

```
#only(1)[Ann] #only(2)[Bob] #only(3)[Christopher] likes #only(1)
[chocolate] #only(2)[strawberry] #only(3)[vanilla] ice cream.
```

Bob likes strawberry ice cream.

But it is hard to see what piece of text actually changes because everything moves around. Better:

```
#alternatives[Ann][Bob][Christopher] likes #alternatives[chocolate]
[strawberry][vanilla] ice cream.
```

Bob likes strawberry ice cream.

#alternatives: Substituting content

You might be tempted to try

```
#only(1)[Ann] #only(2)[Bob] #only(3)[Christopher] likes #only(1)
[chocolate] #only(2)[strawberry] #only(3)[vanilla] ice cream.
```

Christopher likes vanilla ice cream.

But it is hard to see what piece of text actually changes because everything moves around. Better:

```
#alternatives[Ann][Bob][Christopher] likes #alternatives[chocolate]
[strawberry][vanilla] ice cream.
```

Christopher likes vanilla ice cream.

#one-by-one: An alternative for #pause

`#alternatives` is to `#only` what `#one-by-one` is to `#uncover`.

`#one-by-one` behaves similar to using `#pause` but you can additionally state when uncovering should start.

```
#one-by-one(start: 2)[one ][by ][one]
```

`start` can also be omitted, then it starts with the first subside:

```
#one-by-one[one ][by ][one]
```

one

#one-by-one: An alternative for #pause

`#alternatives` is to `#only` what `#one-by-one` is to `#uncover`.

`#one-by-one` behaves similar to using `#pause` but you can additionally state when uncovering should start.

```
#one-by-one(start: 2)[one ][by ][one]
```

one

`start` can also be omitted, then it starts with the first subside:

```
#one-by-one[one ][by ][one]
```

one by

#one-by-one: An alternative for #pause

`#alternatives` is to `#only` what `#one-by-one` is to `#uncover`.

`#one-by-one` behaves similar to using `#pause` but you can additionally state when uncovering should start.

```
#one-by-one(start: 2)[one ][by ][one]
```

one by

`start` can also be omitted, then it starts with the first subside:

```
#one-by-one[one ][by ][one]
```

one by one

#one-by-one: An alternative for #pause

`#alternatives` is to `#only` what `#one-by-one` is to `#uncover`.

`#one-by-one` behaves similar to using `#pause` but you can additionally state when uncovering should start.

```
#one-by-one(start: 2)[one ][by ][one]
```

one by one

`start` can also be omitted, then it starts with the first subside:

```
#one-by-one[one ][by ][one]
```

one by one

#line-by-line: syntactic sugar for #one-by-one

Sometimes it is convenient to write the different contents to uncover one at a time in subsequent lines.

This comes in especially handy for bullet lists, enumerations, and term lists.

```
#line-by-line(start: 2)[
    - first
    - second
    - third
]
```

`start` is again optional and defaults to `1`.

#line-by-line: syntactic sugar for #one-by-one

Sometimes it is convenient to write the different contents to uncover one at a time in subsequent lines.

This comes in especially handy for bullet lists, enumerations, and term lists.

```
#line-by-line(start: 2)[
    - first
    - second
    - third
]
```

- first

`start` is again optional and defaults to `1`.

#line-by-line: syntactic sugar for #one-by-one

Sometimes it is convenient to write the different contents to uncover one at a time in subsequent lines.

This comes in especially handy for bullet lists, enumerations, and term lists.

```
#line-by-line(start: 2)[
    - first
    - second
    - third
]
```

- first
- second

`start` is again optional and defaults to `1`.

#line-by-line: syntactic sugar for #one-by-one

Sometimes it is convenient to write the different contents to uncover one at a time in subsequent lines.

This comes in especially handy for bullet lists, enumerations, and term lists.

```
#line-by-line(start: 2)[
    - first
    - second
    - third
]
```

- first
- second
- third

`start` is again optional and defaults to `1`.

Different ways of covering content

When content is covered, it is completely invisible by default.

However, you can also just display it in light gray by using the `mode` argument with the value `"transparent"`:

Covered content is then displayed differently.

Every `uncover`–based function has an optional `mode` argument:

- `#show: pause(...)`
- `#uncover(...)[...]`
- `#one-by-one(...)[...][...]`
- `#line-by-line(...)[...][...]`

Different ways of covering content

When content is covered, it is completely invisible by default.

However, you can also just display it in light gray by using the `mode` argument with the value `"transparent"`:

Covered content is then displayed differently.

Every `uncover`–based function has an optional `mode` argument:

- `#show: pause(...)`
- `#uncover(...)[...]`
- `#one-by-one(...)[...][...]`
- `#line-by-line(...)[...][...]`

Different ways of covering content

When content is covered, it is completely invisible by default.

However, you can also just display it in light gray by using the `mode` argument with the value `"transparent"`:

Covered content is then displayed differently.

Every `uncover`–based function has an optional `mode` argument:

- `#show: pause(...)`
- `#uncover(...)[...]`
- `#one-by-one(...)[...][...]`
- `#line-by-line(...)[...][...]`

How a slide looks...

... is defined by the theme of the presentation.

This demo uses the default theme.

Because of it, the title slide and the decoration on each slide (with section name, short title, slide number etc.) look the way they do.

Themes can also provide variants, for example ...

… this one!

It’s very minimalist and helps the audience focus on an important point.

Your own theme?

If you want to create your own design for slides, you can define custom themes!

The book explains how to do so.

That's it!

Hopefully you now have some kind of idea what you can do with this template.

Consider giving it a GitHub star or open an issue if you run into bugs or have feature requests.